Jurnal TEKNOINFO, Vol. 14, No. 1, 2020, 1-8, ISSN: 2615-224X
DOI : 10.33365/jti.v14i1.501

PENGAMANAN *CONTAINER ORCHESTRATION* BERBASIS KUBERNETES DI LEMBAGA PENERBANGAN DAN ANTARIKSA NASIONAL (LAPAN)

Arief Indriarto Haris[1)], Rd. Angga Ferianda[2)], Budhi Riyanto[3)], Fajar Iman Nugraha[4)], Januar Abadi[5)]

[1,2,3,4,5)] Lembaga Penerbangan dan Antariksa Nasional (LAPAN)
[1,2,3,4,5)] Jl. Pemuda Persil No.1, Jakarta
Email: [1]arief.indriarto@lapan.go.id, [2]angga.ferianda@lapan.go.id, [3]budhi@lapan.go.id, [4]fajar@lapan.go.id, [5]januar.abadi@lapan.go.id

Abstrak

Melalui Perpres Nomor 39 tahun 2019 tentang Satu Data Indonesia, Lembaga Penerbangan dan Antariksa Nasional (LAPAN) berkomitmen untuk membangun program Satu Data di lingkungan LAPAN, program tersebut dinamakan One Space. Teknologi container platform hadir di LAPAN sebagai pembaruan bagi teknologi sebelumnya (monolithic) dan menjadi salah satu solusi untuk mendukung komitmen pembangunan One Space. Salah satu instrumen didalam teknologi container platform adalah container orchestration, dimana perannya adalah melakukan proses otomasi dan pengelolaan container. Container orchestration dirancang untuk memberikan kelebihan dalam proses deployment terhadap suatu sistem informasi terdistribusi, sehingga menjadi lebih cepat, efisien dan tangkas. Penelitian ini bertujuan untuk mengetahui tindakan pengamanan yang perlu dilakukan pada container orchestration berbasis Kubernetes. Data kualitatif dikumpulkan melalui metode studi literatur, observasi, dan wawancara. Dari hasil pengumpulan data, diperoleh 18 tindakan pengamanan untuk berikutnya akan dinilaikan. Tahap akhir dilakukan proses prioritasi dengan metode MoSCoW melalui sebuah forum group discussion (FGD) bersama dengan para pakar IT dan praktisi IT di lingkungan LAPAN untuk mengetahui tingkat rekomendasi pada masing-masing tindakan pengamanan yang dapat diterapkan di lingkungan LAPAN dengan memperhastikan beberapa parameter. Hasil penelitian ini diperoleh 7 tindakan pengamanan dengan tingkat must have, 10 tindakan pengamanan dengan tingkat should have, dan 1 tindakan pengamanan dengan tingkat could have.

Kata Kunci: *Satu Data; One Space; Container orchestration; Kubernetes*

1. Pendahuluan

Pada era disrupsi seperti saat ini, *data* memegang peranan yang sangat penting di berbagai kalangan, diantaranya masyarakat, bisnis, akademis, hingga penegak hukum dan aparatur pemerintah. Namun dalam praktiknya, *data* masih dikelola secara kurang tepat. Pada *data* pemerintah sendiri, masih terdapat banyak kasus dimana *data* memiliki beragam versi, sehingga sering pula terjadi kontradiktif diantara satu dengan yang lainnya. Melalui Perpres Nomor 39 tahun 2019 tentang Satu Data Indonesia, pemerintah berinisiatif untuk memperbaiki dan meningkatkan kualitas tata kelola *data* pemerintah. Lembaga Penerbangan dan Antariksa Nasional (LAPAN) memiliki komitmen untuk menjalankan program Satu Data di lingkungan LAPAN, dimana program tersebut dinamakan dengan *One Space*. Untuk mewujudkan komitmen tersebut, dibutuhkan suatu sistem dengan teknologi yang mumpuni, memiliki fleksibilitas yang tinggi, mudah untuk diskalakan (*scalability*), dan terjamin ketersediaannya (*availability*).

Pada saat ini, LAPAN menggunakan sistem berbasis teknologi *monolithic* dan sistem berbasis *container platform*. *Monolithic* merupakan suatu sistem tunggal utuh yang didalamnya berisikan sekumpulan prosedur dalam satu kesatuan. Sistem *monolithic* yang dimiliki LAPAN berjalan pada *Virtual Machine* (VM), dengan kata lain setiap sistem *monolithic* membutuhkan VM sebagai wadah untuk melakukan operasional. Sistem jenis ini mudah untuk dibangun, namun disisi lain memiliki beberapa kekurangan diantaranya membutuhkan sumber daya (*resource*) yang besar, sulit untuk diskalakan (*scaling*), dan sulit untuk dilakukan pengembangan sistem.

Dengan pertimbangan tersebut, sistem berbasis *container platform* hadir menjadi salah satu alternatif sekaligus menjadi sebuah pembaruan terhadap teknologi *monolithic* yang saat ini dipergunakan di LAPAN. Pada tahap awal, sistem jenis ini cukup sulit untuk dibangun dikarenakan tingkat kompleksitas yang cukup tinggi. Namun jika sistem ini sudah terbentuk (*established*), maka akan diperoleh beberapa keuntungan diantaranya mudah untuk melakukan pengembangan terhadap sistem jenis ini, karena memiliki tingkat skalabilitas yang cukup tinggi. Selain itu, sistem ini membutuhkan sumber (*resource*) yang lebih kecil dibandingkan dengan sistem berbasis *monolithic*.

Salah satu instrumen dari sistem berbasis *container platform* adalah *container orchestration*. *Container orchestration* berperan dalam melakukan pengelolaan terhadap container secara keseluruhan. Dengan mempertimbangkan peran *container orchestration* yang sangat kritis didalam sistem berbasis teknologi *container platform*, maka diperlukan pengamanan pada sistem *container orchestration* berbasis Kubernetes, untuk dapat

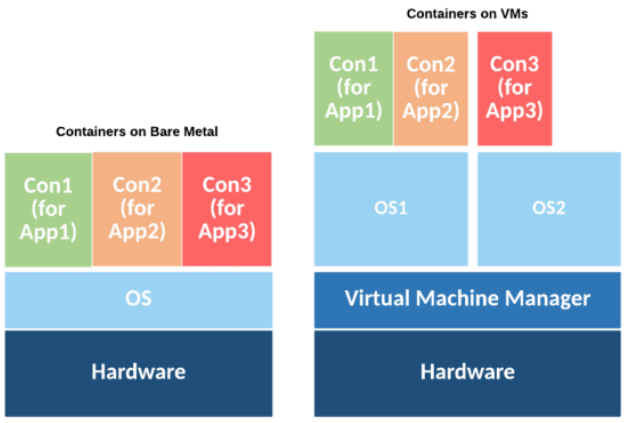
menjaga segala sumber daya informasi yang ada didalamnya agar tetap aman dan tidak disalah gunakan oleh pihak-pihak yang tidak berkepentingan.

1.1. Kubernetes

Kubernetes adalah sebuah *container orchestration open source* untuk menjalankan aplikasi *container*. Kubernetes secara resmi dikembangkan oleh Google, terinspirasi dari pengalaman selama satu dekade dalam proses menjalankan sistem yang *scalable*, *reliable* didalam *container* melalui *application-oriented APIs* [1]. Tetapi Kubernetes lebih dari sekadar mengekspor teknologi yang dikembangkan oleh Google. Kubernetes telah berkembang menjadi sebuah produk komunitas *open source* yang kaya dan bertumbuh. Hal ini menunjukkan bahwa Kubernetes adalah sebuah produk yang cocok tidah hanya untuk kebutuhan perusahaan skala *internet*, tetapi juga untuk para *developer cloud-native* dari semua skala, dari sebuah *cluster* dari komputer Raspberry Pi hingga ke sebuah *warehouse* yang penuh dengan mesin-mesin terbaru. Kubernetes menyediakan perangkat lunak yang dibutuhkan untuk membangun dan menjalankan sistem terdistribusi yang *reliable* dan *scalable* dengan sukses [2]. Dengan Kubernetes, *container* dapat dengan mudah dikembangkan, dihancurkan, dibuat ulang. Dibandingkan dengan *virtual machine*, *container* dapat dijalankan lebih cepat, lebih efisien dan andal [3]. Terdapat banyak hal yang melatarbelakangi mengapa banyak orang memilih untuk menggunakan *container* dan *container API* seperti Kubernetes, diantaranya adalah kecepatan, *scaling* (baik *software* dan tim), meringkaskan infrastruktur, efisiensi [2].

1.2. *Container*

Container dikenal juga sebagai sebuah sistem operasi virtualisasi yang ringan dan menjadi alternatif lain dari virtualisasi berbasis hypervisor. Container membuat instance dengan multiple userspace yang terisolasi diatas OS kernel yang sama. Container menyediakan sebuah abstraksi diatas suatu OS kernel yang memungkinkan untuk menjalankan multiple proses didalam sebuah container yang terisolasi dari container lainnya [4].



Gambar 1. Arsitektur Container [5]

1.3. *Microservices*

Microservices adalah kumpulan *service* kecil, otonom yang berjalan bersama-sama [6]. Arsitektur *microservice* merupakan alternatif arsitektur yang lebih terukur dan lebih fleksibel. Pada arsitektur *microservice*, sistem informasi dirancang untuk terdistribusi dan menyediakan layanan secara lebih fokus dan spesifik. Permasalahan besar akan dipecah menjadi beberapa solusi kecil yang disusun dalam satu *service*, dimana setiap *service* memiliki tanggung jawabnya sendiri. Dengan pendekatan ini, suatu sistem informasi akan terdiri dari beberapa *service* yang dapat dikelola dan didistribusikan secara independen, hal ini akan lebih memudahkan sistem untuk beradaptasi terhadap perubahan kebutuhan [7]. Arsitektur *microservice* tidak terlalu berbeda dengan arsitektur standar aplikasi pada umumnya. Masing-masing dan setiap *microservice* memiliki tiga komponen: sebuah bagian *frontend* (*client-side*), beberapa code pada *backend* dan sebuah cara untuk menyimpan atau mengambil beberapa data yang terkait [8].

2. Penelitian Terkait

Teknologi *container platform* menjadi salah satu teknologi yang saat ini cukup populer dan menjadi solusi alternatif bagi pembagunan suatu sistem informasi. Salah satu instrumen di dalam teknologi *container platform* adalah *container orchestration*, salah satunya berbasis Kubernetes. Kubernetes memiliki beberapa mekanisme dalam pengelolaan *container*, diantaranya adalah *failover* dan *autoscaling container* [9]. [10] dalam penelitiannya membahas tentang strategi *autoscaling* yang saat ini ada di Kubernetes dan mengajukan optimalisasi strategi *autoscaling* yang dapat memecahkan permasalahan *response delay* dalam fase pengembangan. Strategi ini menggunakan kombinasi dari *empirical mode decomposition* dan ARIMA *model* untuk memprediksi beban dari *Pod* dan menyesuaikan jumlah *Pod* untuk melakukan prediksi.

[11] dalam penelitiannya membahas tentang penyediaan sebuah desain terkait infrastruktur *network* yang aman untuk menjalankan Kubernetes di dalam *Data Center*. *Network* dan *security* menjadi aspek dan fokus yang penting untuk diperhatikan. Penelitiannya menyediakan sebuah desain rekomendasi *network* dengan mengacu pada Cisco, VMware, dan Forrester *Zero Trust model* sebagai *security guideline*. Hasil dari penelitiannya berupa rekomendasi yang diajukan dalam bentuk suatu desain sesuai dengan penerapan *Zero Trust* dalam *container network*. Pada penelitian sebelumnya yang dilakukan oleh [11] membahas tentang desain *network* terkait dengan Kubernetes *deployment*, namun tidak membahas terkait pengamanan dari Kubernetes itu sendiri. Penelitian ini bertujuan untuk mengetahui tindakan yang diperlukan untuk melakukan pengamanan pada *container orchestration* berbasis Kubernetes dengan mengacu pada tiga standar *de facto* dari tiga literatur, yaitu Kubernetes *Security*, Kubernetes *Security White Paper*, dan Kubernetes *Deployment* & *Security Patterns*.

3. Metode

3.1. Lokasi Penelitian

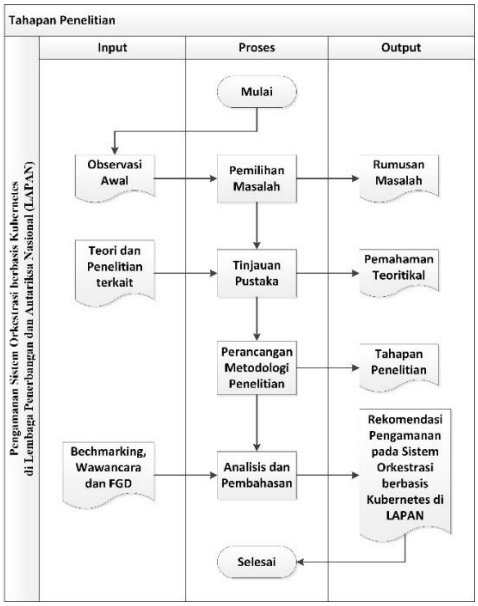
Lokasi pelaksanaan penelitian dilakukan di lingkungan Pusat Teknologi Informasi dan Komunikasi Penerbangan dan Antariksa (Pustikpan) selaku satuan kerja yang diamanahkan oleh LAPAN dalam melakukan pengelolaan Teknologi Informasi dan Komunikasi (TIK) di lingkungan LAPAN.

3.2. Teknik Pengumpulan Data

Dalam melaksanakan kegiatan penelitian ini dilakukan pengumpulan data yang diperoleh dengan metode studi literatur, observasi, dan wawancara. Studi literatur yang dimaksud adalah mengumpulkan data dari buku, jurnal, catatan lapangan dan dokumen teknis. Observasi dilakukan terhadap sistem *container orchestration* secara sistematis. Sedangkan metode wawancara dilakukan untuk mengumpulkan data dengan melibatkan narasumber terkait seperti praktisi IT dan pakar IT.

3.3. Analisis Data

Setelah data kualitatif telah terkumpul, selanjutnya dilakukan analisis data menggunakan metode *scoring* dan MoSCoW. Pada tahap awal, data akan dianalisis menggunakan metode *scoring* untuk mengetahui tingkat rekomendasi dari ketiga sumber literatur terhadap setiap tindakan pengamanan. Selanjutnya dilakukan proses prioritasi dengan metode MoSCoW melalui sebuah *forum group discussion* (FGD) bersama dengan para pakar IT dan praktisi IT di lingkungan LAPAN sehingga dapat dilakukan konstruksi pembahasan hasil penelitian untuk dapat menghasilkan rekomendasi terkait dengan tindakan pengamanan yang tepat terhadap sistem *container orchestration* berbasis Kubernetes di lingkungan LAPAN.



Gambar 2. Metodologi Penelitian

4. Hasil dan Pembahasan

4.1. Hasil

Berikut ini adalah kumpulan tindakan yang diperlukan untuk melakukan pengamanan pada Kubernetes. Adapun data tersebut penulis peroleh dari tiga sumber referensi, yaitu:

1. Kubernetes *Security*, ditulis oleh Liz Rice and Michael Hausenblas (2018);
2. Kubernetes *Security White Paper*, ditulis oleh Stefan Edwards, et al (2019);
3. Kubernetes *Deployment* & *Security Patterns*, ditulis oleh Alex Williams, et al (2018).

Setelah itu dilakukan penilaian, untuk mengetahui tindakan yang paling diprioritaskan dari ketiga sumber tersebut.

Tabel 1. Hasil Pengumpulan *Data* dan *Scoring*

No	Tindakan Pengamanan	Liz Rice dan Michael Hausenblas (2018)	Stefan Edwards, et al (2019)	Alex Williams, et al (2018)	*Score*
1	Menggunakan *authentication module* dan *authorization module* untuk melakukan proses autentikasi dan otorisasi.	√	√	√	3
	Metode:	Untuk autentikasi dapat menggunakan: *static password* atau *token file*, X.509 *certificate*, *OpenID Connect* (OIDC), *Bootstrap token*, *Authenticating proxy, Webhook token authentication*. Untuk otorisasi dapat menggunakan:	Dapat menggunakan: *Role-based access control* (RBAC).	Untuk autentikasi dapat menggunakan: *Client certificate, password, static token, bootstrap token* dan JSON *web token*. Untuk otorisasi dapat menggunakan:	

No	Tindakan Pengamanan	Liz Rice dan Michael Hausenblas (2018)	Stefan Edwards, et al (2019)	Alex Williams, et al (2018)	*Score*
1		*Node authorization*, *Attribute-based access control* (ABAC), *Webhook*, *Role-based access control* (RBAC).		ABAC, RBAC dan *webhook*.	
2	Melakukan *scanning vulnerability* terhadap *container image* dan *host*.	√	-	√	2
	Metode:	Menggunakan *container image scanning tools*, seperti: *OpenScap, Clair, Docker Trusted Registry, Aqua MicroScanner*.	-	Menjalankan *container image scanning* serta mengintegrasikannya dengan *runtime enforcement* dan *remediation capabilities*.	
3	Melakukan *patching* saat ditemukan *vulnerability* pada *container image*.	√	-	√	2
	Metode:	Membangun *container image* baru, lalu melakukan *deploy* kembali berdasarkan *image* baru tersebut. Menerapkan CI/CD.	-	Menerapkan CI/CD.	
4	Menggunakan *private registry* untuk image *storage*.	√	-	√	2
	Metode:	Menggunakan *private registry* untuk mempermudah kontrol terhadap *permission*, baik *read* atau *write* terhadap *image* dan membatasi *network access ke private registry*, seperti menggunakan *firewall*.	-	Menggunakan *private registry*.	
5	Memastikan bahwa hanya *correct version* yang berjalan di *container image*.	√	-	-	1
	Metode:	Menggunakan semantik *version* untuk melakukan *versioning/tag* ke *image*.	-	-	
6	Memastikan *version* yang di-*pull* dari *image registry* adalah asli (*genuine*).	√	-	-	1
	Metode:	Menggunakan *tools* untuk memastikan integritas *image* dan *supply chain*, seperti: Notary *Project*, Grafeas, in-toto.	-	-	
7	Memperkecil *image* agar memperkecil *attack surface*.	√	-	-	1

	Metode:	Meminimalisir jumlah	-	-	
No	**Tindakan Pengamanan**	**Liz Rice dan Michael Hausenblas (2018)**	**Stefan Edwards, et al (2019)**	**Alex Williams, et al (2018)**	***Score***
7		*code* yang terdapat di dalam *image*, hindari penggunaan SSHD, tidak memasukkan utilitas *application code*, seperti perintah cat atau yang lainnya.			
8	Menjalankan *container* dengan *least-privilege*.	√	-	√	2
	Metode:	Menjalankan *container* dengan *non-root user*, pembatasan *mounting host volume*, menonaktifkan *AllowPrivilegeEscalation*.	-	Menerapkan *least-privilege access*, RBAC, atau *access control* yang lainnya untuk mencegah *malicious-user*.	
9	Menjalankan fitur *admission control* uintuk mengaktifkan fitur tambahan dari *access control*.	√	√	√	3
	Metode:	Menggunakan *AlwaysPullImages*, *DenyEscalatingExec*, *PodSecurityPolicy*, *LimitRange*, *ResourceQuota*, *NodeRestriction*.	Menggunakan *podsecuritypolicy*	Menggunakan *podsecuritypolicy* dan *denyescalatingexec*.	
10	Mengaktifkan *runtime resource isolation*.	√	-	√	2
	Metode:	Menggunakan *ResourceQuota*	-	Menggunakan *Namespaces* dan *ResourceQuota*	
11	Menggunakan *network policies* untuk membatasi dan melakukan kontrol terhadap komunikasi dan akses antar *pod* maupun ke *service*.	√	√	√	3
	Metode:	Menggunakan *NetworkPolicy*.	Menggunakan *NetworkPolicy*.	Menggunakan *NetworkPolicy*.	
12	Melakukan enkripsi terhadap komunikasi antar *node*.	√	√	√	3
	Metode:	Menggunakan TLS.	Menggunakan TLS.	Menggunakan TLS.	
13	Mengamankan komunikasi ke API *server*.	√	√	√	3
	Metode:	Menggunakan TLS dan RBAC.	Menggunakan TLS.	Menggunakan TLS.	
14	Melakukan rotasi Kubelet *certificate* secara otomatis.	√	√	-	2
	Metode:	Menggunakan *--rotate-certificates flag*.	Menggunakan AES-GCM.	-	

15	Membangun koneksi yang aman ke etcd.	√	√	-	2
No	**Tindakan Pengamanan**	**Liz Rice dan Michael Hausenblas (2018)**	**Stefan Edwards, et al (2019)**	**Alex Williams, et al (2018)**	***Score***
15	**Metode:**	Menggunakan HTTPS	Menggunakan HTTPS	-	
16	Mengamankan Kubernetes *Dashboard*.	√	-	-	1
	Metode:	Hanya mengijinkan akses yang terautentikasi, menggunakan RBAC, pastikan Kubernetes *Dashboard* hanya memiliki minimal *permission*, jangan membuka akses ke *public* (*internet*).	-	-	
17	Melakukan enkripsi terhadap *secret store* (etcd).	√	√	-	2
	Metode:	Menggunakan AES-CBC, AES-GCM, *Secretbox*, KMS untuk mengenkripsi at *rest configuration*. Menggunakan 32 *byte random* key dan base64 *encoded*.	Menggunakan AES-CBC, AES-GCM, *Secretbox*, KMS.	-	
18	Melakukan *monitoring*, *alerting* dan *auditing* terhadap Kubernetes *cluster*.	√	√	√	3
	Metode:	Menggunakan Prometheus.	-	Menggunakan fitur *Audit Logger*.	

Dari hasil pengumpulan data tersebut, didapatkan 18 tindakan pengamanan, dimana dari setiap tindakan dilakukan penilaian untuk mengetahui tingkat rekomendasi dari ketiga sumber tersebut. Berikutnya dilakukan pengelompokan berdasarkan pada hasil dari penilaian, dapat dilihat pada Tabel 2.

Tabel 2. Hasil Pengelompokan *Data* dan *Scoring*

No	Tindakan Pengamanan	Score
1	Menggunakan *authentication module* dan *authorization module* untuk melakukan proses autentikasi dan otorisasi.	3
	Menjalankan fitur *admission control* uintuk mengaktifkan fitur tambahan dari *access control*.	
	Menggunakan *network policies* untuk membatasi dan melakukan kontrol terhadap komunikasi dan akses antar *pod* maupun ke *service*.	
	Melakukan enkripsi terhadap komunikasi antar *node*.	
	Mengamankan komunikasi ke API *server*.	
	Melakukan *monitoring*, *alerting* dan *auditing* terhadap Kubernetes *cluster*.	
2	Melakukan *scanning vulnerability* terhadap *container image* dan *host*.	2
	Melakukan *patching* saat ditemukan *vulnerability* pada *container image*.	
	Menggunakan *private registry* untuk *image storage*.	
	Menjalankan *container* dengan *least-privilege*.	
	Mengaktifkan *runtime resource isolation*.	
	Melakukan rotasi Kubelet *certificate* secara otomatis.	
	Membangun koneksi yang aman ke etcd.	
	Melakukan enkripsi terhadap *secret store* (etcd).	
3	Memastikan bahwa hanya *correct version* yang berjalan di *container image*.	1
	Memastikan *version* yang ditarik dari *image registry* adalah asli (*genuine*).	
	Memperkecil *image* agar memperkecil *attack surface*.	

	Mengamankan Kubernetes *Dashboard*.	

4.2. Pembahasan

Dari hasil prioritasi berdasarkan penilaian terhadap 18 tindakan pengamanan tersebut, dilakukan proses prioritasi menggunakan metode MoSCoW. Metode ini digunakan untuk mengetahui prioritasi terhadap masing-masing tindakan pengamanan melalui sebuah FGD dengan para pakar IT dan praktisi IT di lingkungan LAPAN. Proses pengukuran diprioritaskan kedalam empat kelompok kategori yaitu *must have*, *should have*, *could have*, *won't have*. Parameter pengukuran yang diperhatikan dalam menentukan prioritasi menggunakan metode MoSCoW antara lain kemudahan dalam penerapan (*applicable*), urgensi dari setiap tindakan pengamanan (*urgency*), dan tingkat pemenuhan dari setiap tindakan pengamanan terhadap kebijakan atau peraturan yang berlaku (*compliance*).

Tabel 3. Hasil Analisis Data Menggunakan Metode MoSCoW

No	Tindakan Pengamanan	Prioritasi
1	Menggunakan *authentication module* dan *authorization module* untuk melakukan proses autentikasi dan otorisasi.	*Must have*
2	Menjalankan fitur *admission control* uintuk mengaktifkan fitur tambahan dari *access control*.	*Should have*
3	Menggunakan *network policies* untuk membatasi dan melakukan kontrol terhadap komunikasi dan akses antar *pod* maupun ke *service*.	*Must have*
4	Melakukan enkripsi terhadap komunikasi antar *node*.	*Must have*
5	Mengamankan komunikasi ke API *server*.	*Must have*
6	Melakukan *monitoring, alerting* dan *auditing* terhadap Kubernetes *cluster*.	*Must have*
7	Melakukan *scanning vulnerability* terhadap *container image* dan *host*.	*Must have*
8	Melakukan *patching* saat ditemukan *vulnerability* pada *container image*.	*Must have*
9	Menggunakan *private registry* untuk *image storage*.	*Should have*
10	Menjalankan *container* dengan *least-privilege*.	*Should have*
11	Mengaktifkan *runtime resource isolation*.	*Should have*
12	Melakukan rotasi Kubelet *certificate* secara otomatis.	*Should have*
13	Membangun koneksi yang aman ke etcd.	*Should have*
14	Melakukan enkripsi terhadap *secret store* (etcd).	*Could have*
15	Memastikan bahwa hanya *correct version* yang berjalan di *container image*.	*Should have*
16	Memastikan *version* yang di-*pull* dari *image registry* adalah asli (*genuine*).	*Should have*
17	Memperkecil *image* agar memperkecil *attack surface*.	*Should have*
18	Mengamankan Kubernetes *Dashboard*.	*Should have*

Dari hasil prioritasi dari setiap tindakan pengamanan yang dilakukan dengan metode MoSCoW dalam sebuah FGD dengan para pakar IT dan praktisi IT di lingkungan LAPAN dengan mempertimbangkan beberapa parameter yang telah disebutkan sebelumnya, didapatkan 7 tindakan pengamanan pada tingkat *must have*, 10 tindakan pengamanan pada tingkat *should have*, dan 1 tindakan pengamanan pada tingkat *could have*.

5. Kesimpulan

Dari hasil penelitian ini diperoleh 7 tindakan pengamanan dengan tingkat *must have*, 10 tindakan pengamanan dengan tingkat *should have*, dan 1 tindakan pengamanan dengan tingkat *could have*. Tindakan pengamanan dengan tingkat must have merupakan suatu kewajiban (*mandatory*) yang harus dilakukan dan menjadi pengamanan dasar yang perlu tersedia dalam suatu sistem *container orchestration* berbasis Kubernetes di lingkungan LAPAN. Tindakan pengamanan dengan tingkat *should have*, lebih disarankan diterapkan untuk mendukung tindakan pengamanan pada tingkat *must have* agar dapat menghasilkan pengamanan yang lebih optimal. Sementara untuk tindakan pengamanan dengan tingkat *could have* adalah tindakan pengamanan yang dapat dilakukan jika tindakan *must have* dan *should have* sudah diterapkan, dengan tetap mempertimbangkan faktor-faktor lain yang ada di lingkungan LAPAN. Untuk kegiatan penelitian berikutnya, disarankan untuk memfokuskan penelitian pada metode dari setiap tindakan pengamanan. Adapun tujuannya adalah untuk mengetahui metode terbaik yang dapat diterapkan dari setiap tindakan pengamanan.

6. Ucapan Terima Kasih

Terima kasih penulis sampaikan kepada seluruh tim Pusat Teknologi Informasi dan Komunikasi Penerbangan dan Antariksa yang sudah memfasilitasi, turut mendukung, dan memberikan masukan sehingga kegiatan penelitian ini dapat berjalan sesuai dengan yang diharapkan.

7. Daftar Pustaka

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[2] K. Hightower, B. Burns, and J. Beda, *Kubernetes Up & Running*. California: O'Reilly Media, 2017.

[3] M. Moilanen and others, "Deploying an application using Docker and Kubernetes," Oulu University of Applied Sciences, 2018.

[4] S. K. Mohanty, "Evaluation of Serverless Computing Frameworks Based on Kubernetes," Aalto University, 2018.

[5] M. Souppaya, J. Morello, and K Scarfone, "Application Container Security Guide," *Nvlpubs.Nist.Gov*, vol. 1, p. 63, 2017.

[6] S. Newman, *Building Microservices*. O'Reilly Media, 2017.

[7] G. Munawar and A. Hodijah, "Analisis Model Arsitektur Microservice Pada Sistem Informasi DPLK," *Sink. (Jurnal Penelit. Tek. Inform.*, vol. 3, 2018.

[8] S. J. Fowler, *Production-Ready Microservices*. California: O'Reilly Media, 2017.

[9] Y. T. Sumbogo, M. Data, and R. A. Siregar, "Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes," *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 2, no. 12, pp. 6849–6854, 2018.

[10] A. Zhao, Q. Huang, Y. Huang, L. Zou, Z. Chen, and J. Song, "Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 569, no. 5, 2019.

[11] N. Surantha and F. Ivan, "Secure Kubernetes Networking Design Based on Zero Trust Model: A Case Study of Financial Service Enterprise in Indonesia," *Adv. Intell. Syst. Comput.*, vol. 994, no. July 2019, pp. 348–361, 2020.